Bài 1:

**Cấu trúc Node (Nút)**

struct Node {

int data; *// 4 bytes - lưu giá trị số nguyên*

Node\* next; *// 4/8 bytes - con trỏ trỏ tới node kế tiếp*

};

**2. Cấu trúc LinkedList**

struct LinkedList {

Node\* head; *// Con trỏ quản lý node đầu tiên*

};

**QUÁ TRÌNH THỰC THI CHI TIẾT**

**Bước 1: Khởi tạo chương trình**

int main() {

LinkedList list; *// Tạo biến list trong stack*

initList(list); *// Gọi hàm khởi tạo*

**Trạng thái bộ nhớ sau khởi tạo:**

Stack:

list.head = NULL (0x00000000)

Heap: (chưa có gì)

**Bước 2: Hàm initList() - Khởi tạo danh sách rỗng**

void initList(LinkedList& list) {

list.head = NULL; *// Gán con trỏ head bằng NULL*

}

**Giải thích:**

* Tham chiếu &list: truyền bằng tham chiếu để thay đổi trực tiếp
* NULL: giá trị đặc biệt表示 danh sách rỗng

**Bước 3: Nhập dữ liệu và gọi addTail()**

**Ví dụ: nhập n = 3, giá trị: 10, 20, 30**

**Lần 1: addTail(list, 10)**

Node\* newNode = new Node; *// Cấp phát bộ nhớ động*

newNode->data = 10;

newNode->next = NULL;

*// Vì list.head == NULL → thực hiện:*

list.head = newNode;

**Trạng thái bộ nhớ:**

Stack: Heap:

list.head = 0x1000 0x1000: [10|NULL]

**Lần 2: addTail(list, 20)**

Node\* temp = list.head; *// temp = 0x1000*

while (temp->next != NULL) *// temp->next = NULL → không vào vòng lặp*

temp->next = newNode; *// Gán next của node 0x1000 thành 0x2000*

**Trạng thái bộ nhớ:**

Stack: Heap:

list.head = 0x1000 0x1000: [10|0x2000] → 0x2000: [20|NULL]

**Lần 3: addTail(list, 30)**

Node\* temp = list.head; *// temp = 0x1000*

while (temp->next != NULL) *// temp->next = 0x2000 ≠ NULL → vào vòng lặp*

temp = temp->next; *// temp = 0x2000*

*// temp hiện tại trỏ đến node cuối (0x2000)*

temp->next = newNode; *// 0x2000->next = 0x3000*

**PHÂN TÍCH TỪNG HÀM CHI TIẾT**

**1. Hàm addTail() - Phân tích từng bước**

void addTail(LinkedList& list, int value) {

*// Bước 1: Tạo node mới*

Node\* newNode = new Node; *// Cấp phát động trong heap*

newNode->data = value; *// Gán giá trị*

newNode->next = NULL; *// Node cuối nên next = NULL*

*// Bước 2: Kiểm tra danh sách rỗng*

if (list.head == NULL) {

list.head = newNode; *// Node mới trở thành head*

return;

}

*// Bước 3: Tìm node cuối cùng*

Node\* temp = list.head; *// Bắt đầu từ head*

while (temp->next != NULL) { *// Duyệt đến khi gặp node cuối*

temp = temp->next; *// Di chuyển đến node kế tiếp*

}

*// Bước 4: Liên kết node cuối với node mới*

temp->next = newNode; *// Hoàn tất thêm vào cuối*

}

**2. Hàm displayList() - Phân tích từng bước**

void displayList(LinkedList list) {

Node\* temp = list.head; *// Tạo con trỏ tạm, trỏ đến head*

*// Duyệt qua từng node*

while (temp != NULL) { *// Lặp đến khi gặp NULL*

cout << temp->data << " "; *// In giá trị node hiện tại*

temp = temp->next; *// Di chuyển đến node kế tiếp*

}

cout << endl;

}

**Quá trình duyệt với ví dụ [10, 20, 30]:**

Lần lặp 1: temp = 0x1000 → data = 10 → temp = 0x2000

Lần lặp 2: temp = 0x2000 → data = 20 → temp = 0x3000

Lần lặp 3: temp = 0x3000 → data = 30 → temp = NULL → KẾT THÚC

**⚡ TỐI ƯU VÀ LƯU Ý QUAN TRỌNG**

**1. Vấn đề hiệu suất addTail()**

**Hiện tại: O(n)** - phải duyệt từ đầu đến cuối mỗi lần thêm

**Cải tiến: Thêm con trỏ**tail

struct LinkedList {

Node\* head;

Node\* tail; *// ← THÊM CON TRỎ CUỐI*

};

void addTailOptimized(LinkedList& list, int value) {

Node\* newNode = new Node{value, NULL};

if (list.head == NULL) {

list.head = list.tail = newNode;

} else {

list.tail->next = newNode;

list.tail = newNode;

}

}

*// Độ phức tạp: O(1) thay vì O(n)*

**2. Quản lý bộ nhớ**

**Vấn đề:** Chương trình hiện tại chưa giải phóng bộ nhớ

**Bổ sung hàm giải phóng:**

void clearList(LinkedList& list) {

Node\* temp = list.head;

while (temp != NULL) {

Node\* nextNode = temp->next; *// Lưu node kế tiếp*

delete temp; *// Giải phóng node hiện tại*

temp = nextNode; *// Di chuyển đến node kế*

}

list.head = NULL; *// Đặt lại head thành NULL*

}

**3. Kiểm tra lỗi**

void addTailSafe(LinkedList& list, int value) {

Node\* newNode = new (nothrow) Node; *// Kiểm tra cấp phát*

if (newNode == NULL) {

cout << "Loi: Khong du bo nho!" << endl;

return;

}

newNode->data = value;

newNode->next = NULL;

*// ... phần còn lại giống addTail()*

}

#include <iostream>

using namespace std;

// 1. Định nghĩa cấu trúc Node

struct Node {

int data; // Dữ liệu lưu trữ (số nguyên)

Node\* next; // Con trỏ trỏ tới node tiếp theo

};

// 2. Định nghĩa cấu trúc Danh sách Liên kết

struct LinkedList {

Node\* head; // Con trỏ trỏ tới node đầu tiên

};

// 3. Hàm khởi tạo danh sách rỗng

void initList(LinkedList& list) {

list.head = NULL; // Gán head bằng NULL → danh sách rỗng

}

// 4. Hàm thêm phần tử vào cuối danh sách

void addTail(LinkedList& list, int value) {

// Tạo node mới

Node\* newNode = new Node;

newNode->data = value;

newNode->next = NULL;

// Trường hợp danh sách rỗng

if (list.head == NULL) {

list.head = newNode;

}

// Trường hợp danh sách có phần tử

else {

Node\* temp = list.head;

// Duyệt đến node cuối cùng

while (temp->next != NULL) {

temp = temp->next;

}

// Liên kết node cuối với node mới

temp->next = newNode;

}

}

// 5. Hàm hiển thị danh sách

void displayList(LinkedList list) {

Node\* temp = list.head; // Bắt đầu từ node đầu tiên

// Duyệt qua tất cả các node

while (temp != NULL) {

cout << temp->data << " "; // In giá trị

temp = temp->next; // Di chuyển đến node tiếp theo

}

cout << endl;

}

// 6. Hàm main - Chương trình chính

int main() {

LinkedList list; // Khai báo danh sách

initList(list); // Khởi tạo danh sách rỗng

int n;

cout << "Nhap so luong phan tu n: ";

cin >> n;

cout << "Nhap " << n << " so nguyen: ";

for (int i = 0; i < n; i++) {

int value;

cin >> value;

addTail(list, value); // Thêm từng số vào danh sách

}

cout << "Danh sach vua nhap: ";

displayList(list); // Hiển thị danh sách

return 0;

}

**Code C/C++ hoàn chỉnh**

#include <iostream>

using namespace std;

// 1. ĐỊNH NGHĨA CẤU TRÚC NODE

struct Node {

int data; // Dữ liệu lưu trữ

Node\* next; // Con trỏ đến node kế tiếp

// Constructor để dễ dàng khởi tạo

Node(int val) : data(val), next(nullptr) {}

};

// 2. ĐỊNH NGHĨA CẤU TRÚC DANH SÁCH LIÊN KẾT

struct LinkedList {

Node\* head; // Con trỏ đến node đầu tiên

// Constructor

LinkedList() : head(nullptr) {}

};

// 3. HÀM KHỞI TẠO DANH SÁCH

void initList(LinkedList& list) {

list.head = nullptr; // Danh sách rỗng

}

// 4. HÀM THÊM PHẦN TỬ VÀO CUỐI DANH SÁCH

void addTail(LinkedList& list, int value) {

// Tạo node mới

Node\* newNode = new Node(value);

// Trường hợp danh sách rỗng

if (list.head == nullptr) {

list.head = newNode;

return;

}

// Trường hợp danh sách không rỗng

// Tìm node cuối cùng

Node\* temp = list.head;

while (temp->next != nullptr) {

temp = temp->next;

}

// Liên kết node cuối với node mới

temp->next = newNode;

}

// 5. HÀM HIỂN THỊ DANH SÁCH

void displayList(const LinkedList& list) {

// Kiểm tra danh sách rỗng

if (list.head == nullptr) {

cout << "Danh sach rong!" << endl;

return;

}

// Duyệt và hiển thị danh sách

Node\* temp = list.head;

cout << "Danh sach: ";

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

// 6. HÀM GIẢI PHÓNG BỘ NHỚ

void clearList(LinkedList& list) {

Node\* current = list.head;

while (current != nullptr) {

Node\* nextNode = current->next; // Lưu node kế tiếp

delete current; // Giải phóng node hiện tại

current = nextNode; // Di chuyển đến node kế

}

list.head = nullptr; // Đặt lại danh sách rỗng

}

// 7. HÀM CHÍNH

int main() {

LinkedList list; // Khai báo danh sách

initList(list); // Khởi tạo danh sách rỗng

int n;

// Nhập số lượng phần tử với kiểm tra

do {

cout << "Nhap so luong phan tu n (n > 0): ";

cin >> n;

if (n <= 0) {

cout << "So luong phan tu phai lon hon 0!" << endl;

}

} while (n <= 0);

// Nhập n số nguyên và thêm vào danh sách

cout << "Nhap " << n << " so nguyen: ";

for (int i = 0; i < n; i++) {

int value;

cin >> value;

addTail(list, value);

}

// Hiển thị danh sách

displayList(list);

// Giải phóng bộ nhớ trước khi kết thúc

clearList(list);

return 0;

}

**Bài Tập 2: Danh Sách Liên Kết Đơn Lưu Trữ Hàm Mũ Một Biến**

**Mô Tả Bài Toán**

Tạo danh sách liên kết đơn để lưu trữ **hệ số** và **số mũ** của một đa thức một biến. Nhập vào một biểu thức đa thức và hiển thị biểu thức đó dưới dạng chuẩn.

**Yêu Cầu Chi Tiết**

* Mỗi phần tử trong danh sách lưu: **hệ số (coefficient)** và **số mũ (exponent)**
* Nhập đa thức từ người dùng
* Hiển thị đa thức dưới dạng chuẩn (ví dụ: 3x^2 + 2x + 1)

**Cấu Trúc Dữ Liệu**

**1. Cấu trúc Term (Số hạng)**

struct Term {

double heSo; *// Hệ số (có thể là số thực)*

int soMu; *// Số mũ (số nguyên)*

Term\* next; *// Con trỏ đến số hạng kế tiếp*

};

**2. Cấu trúc DaThuc (Đa thức)**

struct DaThuc {

Term\* head; *// Con trỏ đến số hạng đầu tiên*

};

**Code C/C++ đầy đủ**

#include <iostream>

#include <iomanip>

using namespace std;

// 1. CẤU TRÚC SỐ HẠNG

struct Term {

double heSo; // Hệ số

int soMu; // Số mũ

Term\* next; // Con trỏ đến số hạng kế tiếp

// Constructor

Term(double hs = 0, int sm = 0) : heSo(hs), soMu(sm), next(nullptr) {}

};

// 2. CẤU TRÚC ĐA THỨC

struct DaThuc {

Term\* head; // Con trỏ đến số hạng đầu tiên

// Constructor

DaThuc() : head(nullptr) {}

};

// 3. HÀM KHỞI TẠO ĐA THỨC RỖNG

void initDaThuc(DaThuc& dt) {

dt.head = nullptr;

}

// 4. HÀM THÊM SỐ HẠNG VÀO ĐA THỨC (sắp xếp giảm dần theo số mũ)

void themSoHang(DaThuc& dt, double heSo, int soMu) {

// Bỏ qua số hạng có hệ số = 0

if (heSo == 0) return;

Term\* newTerm = new Term(heSo, soMu);

// Trường hợp đa thức rỗng hoặc thêm vào đầu

if (dt.head == nullptr || soMu > dt.head->soMu) {

newTerm->next = dt.head;

dt.head = newTerm;

return;

}

// Tìm vị trí thích hợp (sắp xếp giảm dần theo số mũ)

Term\* current = dt.head;

while (current->next != nullptr && current->next->soMu > soMu) {

current = current->next;

}

// Kiểm tra nếu số mũ đã tồn tại thì cộng hệ số

if (current->soMu == soMu) {

current->heSo += heSo;

delete newTerm; // Không cần tạo mới

// Nếu hệ số trở thành 0 thì xóa số hạng

if (current->heSo == 0) {

// Xử lý xóa node (cần implement thêm)

}

} else {

// Chèn vào vị trí thích hợp

newTerm->next = current->next;

current->next = newTerm;

}

}

// 5. HÀM NHẬP ĐA THỨC TỪ NGƯỜI DÙNG

void nhapDaThuc(DaThuc& dt) {

int n;

cout << "Nhap so luong so hang cua da thuc: ";

cin >> n;

for (int i = 0; i < n; i++) {

double heSo;

int soMu;

cout << "Nhap so hang thu " << (i + 1) << ":" << endl;

cout << " He so: ";

cin >> heSo;

cout << " So mu: ";

cin >> soMu;

themSoHang(dt, heSo, soMu);

}

}

// 6. HÀM HIỂN THỊ ĐA THỨC

void hienThiDaThuc(const DaThuc& dt) {

if (dt.head == nullptr) {

cout << "0";

return;

}

Term\* current = dt.head;

bool firstTerm = true;

while (current != nullptr) {

// Xử lý dấu

if (!firstTerm) {

if (current->heSo > 0) {

cout << " + ";

} else {

cout << " - ";

}

} else {

if (current->heSo < 0) {

cout << "-";

}

}

// Hiển thị hệ số (lấy giá trị tuyệt đối)

double absHeSo = abs(current->heSo);

// Hiển thị số hạng

if (current->soMu == 0) {

// Số hạng hằng số

cout << absHeSo;

} else if (current->soMu == 1) {

// Số hạng bậc 1

if (absHeSo == 1) {

cout << "x";

} else {

cout << absHeSo << "x";

}

} else {

// Số hạng bậc cao

if (absHeSo == 1) {

cout << "x^" << current->soMu;

} else {

cout << absHeSo << "x^" << current->soMu;

}

}

firstTerm = false;

current = current->next;

}

cout << endl;

}

// 7. HÀM GIẢI PHÓNG BỘ NHỚ

void xoaDaThuc(DaThuc& dt) {

Term\* current = dt.head;

while (current != nullptr) {

Term\* nextTerm = current->next;

delete current;

current = nextTerm;

}

dt.head = nullptr;

}

// 8. HÀM NHẬP ĐA THỨC TỪ CHUỖI (NÂNG CAO)

void nhapDaThucTuChuoi(DaThuc& dt, const string& chuoi) {

// Giả sử chuỗi có dạng: "3x^2 + 2x + 1"

// Trong thực tế cần phân tích chuỗi phức tạp hơn

// Ở đây demo đơn giản

cout << "Nhap da thuc theo dang: ax^n + bx^m + ... + c" << endl;

cout << "Vi du: 3x^2 + 2x + 1" << endl;

int n;

cout << "Nhap so luong so hang: ";

cin >> n;

for (int i = 0; i < n; i++) {

double heSo;

int soMu;

cout << "So hang " << (i + 1) << " - He so: ";

cin >> heSo;

cout << "So hang " << (i + 1) << " - So mu: ";

cin >> soMu;

themSoHang(dt, heSo, soMu);

}

}

// 9. HÀM CHÍNH

int main() {

DaThuc dt;

initDaThuc(dt);

int choice;

do {

cout << "\n QUAN LY DA THUC " << endl;

cout << "1. Nhap da thuc" << endl;

cout << "2. Hien thi da thuc" << endl;

cout << "3. Thoat" << endl;

cout << "Chon chuc nang: ";

cin >> choice;

switch (choice) {

case 1:

xoaDaThuc(dt); // Xóa đa thức cũ trước khi nhập mới

nhapDaThuc(dt);

break;

case 2:

cout << "Da thuc: ";

hienThiDaThuc(dt);

break;

case 3:

cout << "Thoat chuong trinh." << endl;

break;

default:

cout << "Chuc nang khong hop le!" << endl;

}

} while (choice != 3);

// Giải phóng bộ nhớ trước khi thoát

xoaDaThuc(dt);

return 0;

}

**Phân Tích Giải Thuật Chi Tiết**

**1. Giải Thuật themSoHang()**

**Mục đích:** Thêm số hạng mới vào đa thức, duy trì thứ tự giảm dần theo số mũ

**Các bước:**

1. KIỂM TRA hệ số = 0 → bỏ qua

2. TẠO node mới chứa hệ số và số mũ

3. XỬ LÝ các trường hợp:

- Đa thức rỗng → thêm vào đầu

- Số mũ lớn hơn head → thêm vào đầu

- Ngược lại → tìm vị trí thích hợp

4. KIỂM TRA trùng số mũ → cộng dồn hệ số

5. CHÈN vào vị trí phù hợp

**Ví dụ:**

Đa thức hiện tại: 3x^2 + 2x

Thêm: 4x^3 → Kết quả: 4x^3 + 3x^2 + 2x

Thêm: 1x^2 → Kết quả: 4x^3 + 4x^2 + 2x (cộng dồn hệ số)

**2. Giải Thuật hienThiDaThuc()**

**Mục đích:** Hiển thị đa thức dưới dạng chuẩn

**Quy tắc hiển thị:**

- Số hạng đầu: không có dấu + (trừ khi âm)

- Các số hạng sau: có dấu + hoặc -

- Hệ số 1: không hiển thị (trừ hằng số)

- Số mũ 0: chỉ hiển thị hệ số

- Số mũ 1: hiển thị x (không hiển thị ^1)

**Ví dụ:**

[3x^2, -2x, 1] → "3x^2 - 2x + 1"

[1x^3, -1x, 5] → "x^3 - x + 5"

**Ví Dụ Minh Họa**

**Input:**

Số lượng số hạng: 3

Số hạng 1: Hệ số = 3, Số mũ = 2

Số hạng 2: Hệ số = -2, Số mũ = 1

Số hạng 3: Hệ số = 1, Số mũ = 0

**Quá trình xử lý:**

**Bước 1:** Thêm 3x^2

Đa thức: 3x^2

**Bước 2:** Thêm -2x^1

Đa thức: 3x^2 - 2x

**Bước 3:** Thêm 1x^0

Đa thức: 3x^2 - 2x + 1

**Output:**

Da thuc: 3x^2 - 2x + 1

**Phiên Bản Nâng Cao**

**Nhập đa thức từ chuỗi (ý tưởng):**

void nhapTuChuoiPhanTich(DaThuc& dt, const string& str) {

*// Phân tích chuỗi như: "3x^2 + 2x - 5"*

*// Sử dụng các thuật toán xử lý chuỗi*

*// Tách các số hạng, phân tích hệ số và số mũ*

}

**Bài tập 3 — Khởi tạo & thao tác cơ bản với danh sách liên kết đơn**

**Thuật Toán Tổng Quát**

Bước 1: KHỞI TẠO

head = NULL

Bước 2: NHẬP DỮ LIỆU

Đọc n

Với mỗi i từ 1 đến n:

Đọc giá trị x

Gọi push\_back(&head, x)

Bước 3: HIỂN THỊ

Gọi print\_list(head)

Bước 4: ĐẾM PHẦN TỬ

count = size(head)

In count

Bước 5: TÌM KIẾM

Đọc giá trị x cần tìm

result = find\_first(head, x)

Nếu result != NULL → "FOUND"

Ngược lại → "NOT FOUND"

Bước 6: DỌN DẸP

Gọi clear\_list(&head)

**Input:**

5

10 20 30 40 50

30

**Output:**

Danh sach: 10 20 30 40 50

So phan tu: 5

FOUND: 30

**Input:**

4

5 15 25 35

10

**Output:**

Danh sach: 5 15 25 35

So phan tu: 4

NOT FOUND

**Code C/C++ đầy đủ**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

// Tạo node mới

Node\* make\_node(int x){

Node \*p = (Node \*)malloc(sizeof(Node));

if(p == NULL){

printf("Khong du bo nho!\n");

return NULL;

}

p->data = x;

p->next = NULL;

return p;

}

// Thêm vào cuối danh sách

void push\_back(Node \*\*head, int x){

Node \*p = make\_node(x);

if(p == NULL) return;

if(\*head == NULL){

\*head = p;

return;

}

Node \*cur = \*head;

while(cur->next != NULL)

cur = cur->next;

cur->next = p;

}

// Thêm vào đầu danh sách

void push\_front(Node \*\*head, int x){

Node \*p = make\_node(x);

if(p == NULL) return;

p->next = \*head;

\*head = p;

}

// In danh sách

void print\_list(Node \*head){

for(Node \*cur = head; cur != NULL; cur = cur->next)

printf("%d ", cur->data);

printf("\n");

}

// Đếm số phần tử

int size(Node \*head){

int cnt = 0;

for(Node \*c = head; c != NULL; c = c->next)

cnt++;

return cnt;

}

// Tìm giá trị x đầu tiên

Node\* find\_first(Node \*head, int x){

for(Node \*c = head; c != NULL; c = c->next)

if(c->data == x)

return c;

return NULL;

}

// Giải phóng bộ nhớ

void clear\_list(Node \*\*head){

Node \*c = \*head;

while(c != NULL){

Node \*next = c->next;

free(c);

c = next;

}

\*head = NULL;

}

int main(){

Node \*head = NULL;

int n;

// Đọc n

if(scanf("%d", &n) != 1)

return 0;

// Đọc n số và thêm vào cuối

for(int i = 0; i < n; i++){

int x;

scanf("%d", &x);

push\_back(&head, x);

}

// In danh sách

printf("Danh sach: ");

print\_list(head);

// Đếm số phần tử

printf("So phan tu: %d\n", size(head));

// Tìm giá trị x

int x;

printf("Nhap gia tri can tim: ");

scanf("%d", &x);

Node \*result = find\_first(head, x);

if(result != NULL)

printf("FOUND: %d\n", result->data);

else

printf("NOT FOUND\n");

// Giải phóng bộ nhớ

clear\_list(&head);

return 0;

}

**Phân Tích Chi Tiết Các Hàm**

**1. Hàm make\_node(int x)**

c

Node\* make\_node(int x){

Node \*p = (Node \*)malloc(sizeof(Node));

if(p == NULL){

printf("Khong du bo nho!\n");

return NULL;

}

p->data = x;

p->next = NULL;

return p;

}

**Mục đích:** Tạo node mới  
**Các bước:**

* Cấp phát bộ nhớ cho node
* Kiểm tra cấp phát thành công
* Gán giá trị và khởi tạo con trỏ next

**2. Hàm push\_back(Node \*\*head, int x)**

c

void push\_back(Node \*\*head, int x){

Node \*p = make\_node(x);

if(\*head == NULL){

\*head = p;

return;

}

Node \*cur = \*head;

while(cur->next != NULL)

cur = cur->next;

cur->next = p;

}

**Mục đích:** Thêm vào cuối danh sách  
**Các bước:**

* Tạo node mới
* Nếu danh sách rỗng → head trỏ đến node mới
* Ngược lại → duyệt đến cuối và thêm vào

**3. Hàm push\_front(Node \*\*head, int x)**

c

void push\_front(Node \*\*head, int x){

Node \*p = make\_node(x);

p->next = \*head;

\*head = p;

}

**Mục đích:** Thêm vào đầu danh sách  
**Các bước:**

* Tạo node mới
* Node mới trỏ đến head cũ
* Head trỏ đến node mới

**4. Hàm find\_first(Node \*head, int x)**

c

Node\* find\_first(Node \*head, int x){

for(Node \*c = head; c != NULL; c = c->next)

if(c->data == x)

return c;

return NULL;

}

**Mục đích:** Tìm giá trị x đầu tiên  
**Các bước:**

* Duyệt từ đầu đến cuối
* So sánh từng phần tử với x
* Trả về node đầu tiên tìm thấy, hoặc NULL nếu không tìm thấy

**🎯 Ví Dụ Minh Họa**

**Input:**

5

10 20 30 20 40

20

**Quá trình thực hiện:**

**Bước 1: Nhập n = 5**

**Bước 2: Thêm 5 số vào cuối**

10 → 20 → 30 → 20 → 40 → NULL

**Bước 3: In danh sách**

Danh sach: 10 20 30 20 40

**Bước 4: Đếm phần tử**

So phan tu: 5

**Bước 5: Tìm giá trị 20**

Tìm thấy node đầu tiên có giá trị 20

FOUND: 20

**Output:**

Danh sach: 10 20 30 20 40

So phan tu: 5

Nhap gia tri can tim: 20

FOUND: 20

**Bài Tập 4 - Đảo Ngược Danh Sách Liên Kết Đơn (Iterative)**

**Mô Tả Bài Toán**

**Yêu cầu:** Viết hàm reverse(head) đảo ngược danh sách liên kết đơn **in-place** sử dụng **3 con trỏ** (prev, cur, next) mà không dùng mảng phụ hay cấp phát bộ nhớ mới.

**Yêu Cầu Chi Tiết**

1. Thuật toán đảo ngược in-place (không tạo danh sách mới)
2. Chỉ sử dụng 3 con trỏ: prev, cur, next
3. Không dùng mảng phụ hay cấp phát bộ nhớ mới
4. Chỉ thay đổi liên kết giữa các node
5. Hiển thị danh sách trước và sau khi đảo ngược

**Cấu trúc dữ liệu**

typedef struct Node {

int data;

struct Node\* next;

} Node;

**Thuật Toán Đảo Ngược Với 3 Con Trỏ**

**Sơ Đồ Thuật Toán**

Ban đầu: 1 → 2 → 3 → 4 → NULL

Sau đảo: NULL ← 1 ← 2 ← 3 ← 4

**Mô Tả Các Bước**

Bước 0: Khởi tạo

prev = NULL

cur = head

next = NULL

Bước 1: Lặp cho đến khi cur == NULL

1.1. Lưu next = cur->next

1.2. Đảo liên kết: cur->next = prev

1.3. Di chuyển: prev = cur

1.4. Di chuyển: cur = next

Bước 2: Cập nhật head = prev

**Code C/C++ đầy đủ**

#include <stdio.h>

#include <stdlib.h>

// Định nghĩa cấu trúc Node

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Hàm tạo node mới

Node\* make\_node(int x) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Khong du bo nho!\n");

return NULL;

}

newNode->data = x;

newNode->next = NULL;

return newNode;

}

// Hàm thêm vào cuối danh sách

void push\_back(Node\*\* head, int x) {

Node\* newNode = make\_node(x);

if (newNode == NULL) return;

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

// Hàm in danh sách

void print\_list(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

// HÀM ĐẢO NGƯỢC DANH SÁCH (ITERATIVE) - Sử dụng 3 con trỏ

Node\* reverse(Node\* head) {

Node\* prev = NULL; // Con trỏ trước

Node\* cur = head; // Con trỏ hiện tại

Node\* next = NULL; // Con trỏ kế tiếp

// Duyệt qua toàn bộ danh sách

while (cur != NULL) {

// Bước 1: Lưu node kế tiếp

next = cur->next;

// Bước 2: Đảo liên kết

cur->next = prev;

// Bước 3: Di chuyển các con trỏ

prev = cur;

cur = next;

}

// prev bây giờ là node đầu của danh sách đảo ngược

return prev;

}

// Hàm giải phóng bộ nhớ

void clear\_list(Node\*\* head) {

Node\* current = \*head;

while (current != NULL) {

Node\* next = current->next;

free(current);

current = next;

}

\*head = NULL;

}

// Hàm main

int main() {

Node\* head = NULL;

int n, x;

// Nhập số lượng phần tử

printf("Nhap so luong phan tu: ");

scanf("%d", &n);

// Nhập các phần tử

printf("Nhap %d phan tu: ", n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

push\_back(&head, x);

}

// Hiển thị danh sách trước khi đảo

printf("Danh sach truoc khi dao nguoc: ");

print\_list(head);

// Đảo ngược danh sách

head = reverse(head);

// Hiển thị danh sách sau khi đảo

printf("Danh sach sau khi dao nguoc: ");

print\_list(head);

// Giải phóng bộ nhớ

clear\_list(&head);

return 0;

}

**Phân Tích Chi Tiết Thuật Toán Đảo Ngược**

**Hàm reverse - Giải Thích Từng Bước**

Node\* reverse(Node\* head) {

Node\* prev = NULL; *// Node trước (ban đầu là NULL)*

Node\* cur = head; *// Node hiện tại (bắt đầu từ head)*

Node\* next = NULL; *// Node kế tiếp*

while (cur != NULL) {

*// Bước 1: Lưu trữ node kế tiếp trước khi đảo liên kết*

next = cur->next;

*// Bước 2: Đảo liên kết - node hiện tại trỏ về node trước*

cur->next = prev;

*// Bước 3: Di chuyển prev và cur tiến lên*

prev = cur; *// prev di chuyển đến vị trí hiện tại*

cur = next; *// cur di chuyển đến vị trí kế tiếp*

}

return prev; *// prev bây giờ là head mới*

}

**Chạy Chương Trình**

**Input:**

Nhap so luong phan tu: 5

Nhap 5 phan tu: 10 20 30 40 50

**Output:**

Danh sach truoc khi dao nguoc: 10 20 30 40 50

Danh sach sau khi dao nguoc: 50 40 30 20 10

**Input:**

Nhap so luong phan tu: 3

Nhap 3 phan tu: 5 15 25

**Output:**

Danh sach truoc khi dao nguoc: 5 15 25

Danh sach sau khi dao nguoc: 25 15 5

**Bài Tập 5 - Xóa Phần Tử Trùng Và Chèn Có Thứ Tự**

**Cấu Trúc Dữ Liệu**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

**Code C/C++ đầy đủ**

#include <stdio.h>

#include <stdlib.h>

// Định nghĩa cấu trúc Node

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Hàm tạo node mới

Node\* make\_node(int x) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Khong du bo nho!\n");

return NULL;

}

newNode->data = x;

newNode->next = NULL;

return newNode;

}

// Hàm thêm vào cuối danh sách

void push\_back(Node\*\* head, int x) {

Node\* newNode = make\_node(x);

if (newNode == NULL) return;

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

// Hàm in danh sách

void print\_list(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

// Hàm giải phóng bộ nhớ

void clear\_list(Node\*\* head) {

Node\* current = \*head;

while (current != NULL) {

Node\* next = current->next;

free(current);

current = next;

}

\*head = NULL;

}

// PHẦN 1: XÓA PHẦN TỬ TRÙNG

// Hàm xóa phần tử trùng (giữ lại lần xuất hiện đầu tiên)

void remove\_duplicates\_unsorted(Node\* head) {

if (head == NULL) return;

Node\* i = head; // Con trỏ duyệt ngoài

// Duyệt từng phần tử i

while (i != NULL && i->next != NULL) {

Node\* j = i; // Con trỏ duyệt trong, bắt đầu từ i

// Duyệt các phần tử phía sau i

while (j->next != NULL) {

// Nếu tìm thấy phần tử trùng với i

if (j->next->data == i->data) {

Node\* duplicate = j->next; // Node cần xóa

j->next = j->next->next; // Bỏ qua node trùng

free(duplicate); // Giải phóng bộ nhớ

} else {

j = j->next; // Di chuyển đến node kế tiếp

}

}

i = i->next; // Di chuyển đến node kế tiếp

}

}

// PHẦN 2: CHÈN CÓ THỨ TỰ

// Hàm chèn có thứ tự vào danh sách đã sắp xếp tăng dần

void sorted\_insert(Node\*\* head, int x) {

Node\* newNode = make\_node(x);

if (newNode == NULL) return;

// Trường hợp 1: Danh sách rỗng hoặc chèn vào đầu

if (\*head == NULL || x < (\*head)->data) {

newNode->next = \*head;

\*head = newNode;

return;

}

// Trường hợp 2: Tìm vị trí thích hợp để chèn

Node\* current = \*head;

// Duyệt đến node có giá trị LỚN HƠN hoặc BẰNG x

while (current->next != NULL && current->next->data < x) {

current = current->next;

}

// Chèn newNode vào sau current

newNode->next = current->next;

current->next = newNode;

}

// HÀM SẮP XẾP DANH SÁCH

// Hàm sắp xếp danh sách tăng dần (sử dụng selection sort)

void sort\_list(Node\*\* head) {

if (\*head == NULL) return;

Node\* i = \*head;

while (i != NULL) {

Node\* minNode = i;

Node\* j = i->next;

// Tìm phần tử nhỏ nhất trong phần chưa sắp xếp

while (j != NULL) {

if (j->data < minNode->data) {

minNode = j;

}

j = j->next;

}

// Đổi chỗ dữ liệu

int temp = i->data;

i->data = minNode->data;

minNode->data = temp;

i = i->next;

}

}

// HÀM MAIN

int main() {

Node\* head = NULL;

int n, x, choice;

printf("Nhap so luong phan tu: ");

scanf("%d", &n);

printf("Nhap %d phan tu: ", n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

push\_back(&head, x);

}

do {

printf("\n MENU CHUC NANG \n");

printf("1. Hien thi danh sach\n");

printf("2. Xoa phan tu trung (keep first)\n");

printf("3. Sap xep danh sach tang dan\n");

printf("4. Chen co thu tu\n");

printf("5. Thoat\n");

printf("Chon chuc nang: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Danh sach hien tai: ");

print\_list(head);

break;

case 2:

remove\_duplicates\_unsorted(head);

printf("Da xoa phan tu trung! Danh sach moi: ");

print\_list(head);

break;

case 3:

sort\_list(&head);

printf("Da sap xep danh sach! Danh sach moi: ");

print\_list(head);

break;

case 4:

printf("Nhap gia tri can chen: ");

scanf("%d", &x);

sorted\_insert(&head, x);

printf("Da chen %d! Danh sach moi: ", x);

print\_list(head);

break;

case 5:

printf("Thoat chuong trinh.\n");

break;

default:

printf("Chuc nang khong hop le!\n");

}

} while (choice != 5);

clear\_list(&head);

return 0;

}

**Thuật Toán remove\_duplicates\_unsorted**

**Mô tả:** Xóa phần tử trùng, giữ lại lần xuất hiện đầu tiên

**Các bước:**

for i từ đầu đến cuối:

for j từ i đến cuối:

if j->next trùng với i->data:

xóa j->next

else:

j di chuyển tiếp

**Thuật Toán sorted\_insert**

**Mô tả:** Chèn giá trị x vào danh sách đã sắp xếp tăng dần

**Các trường hợp:**

Trường hợp 1: Danh sách rỗng hoặc x nhỏ hơn head

→ Chèn vào đầu

Trường hợp 2: Tìm vị trí thích hợp

→ Duyệt đến node có giá trị >= x

→ Chèn vào trước node đó

**Chạy Chương Trình**

**Test Case 1: Xóa phần tử trùng**

Input:

Nhap so luong phan tu: 6

Nhap 6 phan tu: 1 2 1 3 2 4

Chon 2: Xoa phan tu trung

Output: 1 2 3 4

**Test Case 2: Chèn có thứ tự**

Input:

Danh sach: 1 3 5

Chon 4: Chen 2

Output: 1 2 3 5

Chen 4

Output: 1 2 3 4 5

Chen 0

Output: 0 1 2 3 4 5

Chen 6

Output: 0 1 2 3 4 5 6

**Đánh Giá Thuật Toán**

**remove\_duplicates\_unsorted:**

* **Ưu điểm:** Đơn giản, dễ hiểu, không cần thêm bộ nhớ
* **Nhược điểm:** O(n²) - chậm với danh sách lớn
* **Cải tiến:** Có thể dùng hash table để đạt O(n)

**sorted\_insert:**

* **Ưu điểm:** Hiệu quả, duy trì thứ tự
* **Nhược điểm:** Vẫn cần duyệt tìm vị trí
* **Ứng dụng:** Xây dựng danh sách có thứ tự từ đầu

**Bài Tập 6 - Trộn Hai Danh Sách Đã Sắp Xếp**

**Cấu Trúc Dữ Liệu**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

**Code C/C++ đầy đủ**

#include <stdio.h>

#include <stdlib.h>

// Định nghĩa cấu trúc Node

typedef struct Node {

int data;

struct Node\* next;

} Node;

// Hàm tạo node mới

Node\* make\_node(int x) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

if (newNode == NULL) {

printf("Khong du bo nho!\n");

return NULL;

}

newNode->data = x;

newNode->next = NULL;

return newNode;

}

// Hàm thêm vào cuối danh sách

void push\_back(Node\*\* head, int x) {

Node\* newNode = make\_node(x);

if (newNode == NULL) return;

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

// Hàm in danh sách

void print\_list(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

// Hàm giải phóng bộ nhớ

void clear\_list(Node\*\* head) {

Node\* current = \*head;

while (current != NULL) {

Node\* next = current->next;

free(current);

current = next;

}

\*head = NULL;

}

// HÀM TRỘN HAI DANH SÁCH ĐÃ SẮP XẾP

// Hàm trộn hai danh sách đã sắp xếp (không cấp phát node mới)

Node\* merge\_sorted(Node\* a, Node\* b) {

// Tạo node giả (dummy node) để đơn giản hóa logic

Node dummy = {0, NULL};

Node\* tail = &dummy; // Con trỏ tail luôn trỏ đến node cuối của danh sách kết quả

// So sánh và trộn cho đến khi một trong hai danh sách hết

while (a != NULL && b != NULL) {

if (a->data <= b->data) {

// Thêm node từ danh sách a vào kết quả

tail->next = a;

a = a->next;

} else {

// Thêm node từ danh sách b vào kết quả

tail->next = b;

b = b->next;

}

tail = tail->next; // Di chuyển tail đến node cuối mới

}

// Nối phần còn lại của danh sách chưa hết

tail->next = (a != NULL) ? a : b;

// Trả về danh sách kết quả (bỏ qua node giả)

return dummy.next;

}

// HÀM SẮP XẾP DANH SÁCH

// Hàm sắp xếp danh sách tăng dần (sử dụng insertion sort)

void sort\_list(Node\*\* head) {

if (\*head == NULL || (\*head)->next == NULL) return;

Node\* sorted = NULL; // Danh sách đã sắp xếp (ban đầu rỗng)

Node\* current = \*head;

while (current != NULL) {

Node\* next = current->next; // Lưu node kế tiếp

// Chèn current vào danh sách sorted

if (sorted == NULL || current->data <= sorted->data) {

// Chèn vào đầu

current->next = sorted;

sorted = current;

} else {

// Tìm vị trí thích hợp để chèn

Node\* temp = sorted;

while (temp->next != NULL && temp->next->data < current->data) {

temp = temp->next;

}

current->next = temp->next;

temp->next = current;

}

current = next;

}

\*head = sorted;

}

// HÀM KIỂM TRA DANH SÁCH ĐÃ SẮP XẾP

// Hàm kiểm tra danh sách có được sắp xếp tăng dần không

int is\_sorted(Node\* head) {

if (head == NULL || head->next == NULL) return 1;

Node\* current = head;

while (current->next != NULL) {

if (current->data > current->next->data) {

return 0; // Không sắp xếp

}

current = current->next;

}

return 1; // Đã sắp xếp

}

// HÀM MAIN

int main() {

Node\* listA = NULL;

Node\* listB = NULL;

Node\* mergedList = NULL;

int n, m, x;

printf(" TRON HAI DANH SACH DA SAP XEP \n\n");

// Nhập danh sách A

printf("Nhap so luong phan tu cho danh sach A: ");

scanf("%d", &n);

printf("Nhap %d phan tu cho danh sach A: ", n);

for (int i = 0; i < n; i++) {

scanf("%d", &x);

push\_back(&listA, x);

}

// Sắp xếp danh sách A nếu chưa sắp xếp

if (!is\_sorted(listA)) {

printf("Danh sach A chua sap xep. Dang sap xep...\n");

sort\_list(&listA);

}

// Nhập danh sách B

printf("\nNhap so luong phan tu cho danh sach B: ");

scanf("%d", &m);

printf("Nhap %d phan tu cho danh sach B: ", m);

for (int i = 0; i < m; i++) {

scanf("%d", &x);

push\_back(&listB, x);

}

// Sắp xếp danh sách B nếu chưa sắp xếp

if (!is\_sorted(listB)) {

printf("Danh sach B chua sap xep. Dang sap xep...\n");

sort\_list(&listB);

}

// Hiển thị danh sách trước khi trộn

printf("\n TRUOC KHI TRON \n");

printf("Danh sach A (da sap xep): ");

print\_list(listA);

printf("Danh sach B (da sap xep): ");

print\_list(listB);

// Trộn hai danh sách

mergedList = merge\_sorted(listA, listB);

// Hiển thị kết quả sau khi trộn

printf("\n SAU KHI TRON \n");

printf("Danh sach sau khi tron: ");

print\_list(mergedList);

// Kiểm tra danh sách kết quả có được sắp xếp không

if (is\_sorted(mergedList)) {

printf(" Danh sach ket qua da duoc sap xep tang dan.\n");

} else {

printf("✗ Danh sach ket qua chua duoc sap xep.\n");

}

// Xác nhận danh sách gốc đã trống

printf("\n KIEM TRA NODE \n");

printf("Danh sach A sau tron: ");

print\_list(listA); // Nên in rỗng hoặc không xác định

printf("Danh sach B sau tron: ");

print\_list(listB); // Nên in rỗng hoặc không xác định

printf(" Cac node da duoc tai su dung, khong con node roi.\n");

// Chỉ giải phóng danh sách đã trộn (vì các node đã được tái sử dụng)

clear\_list(&mergedList);

printf("\nChuong trinh ket thuc.\n");

return 0;

}

**Phân Tích Chi Tiết Thuật Toán Trộn**

**Hàm merge\_sorted - Giải Thích Từng Bước**

Node\* merge\_sorted(Node\* a, Node\* b) {

Node dummy = {0, NULL}; *// Node giả để đơn giản hóa*

Node\* tail = &dummy; *// Theo dõi node cuối của danh sách kết quả*

while (a != NULL && b != NULL) {

if (a->data <= b->data) {

tail->next = a; *// Thêm node từ a*

a = a->next; *// Di chuyển a*

} else {

tail->next = b; *// Thêm node từ b*

b = b->next; *// Di chuyển b*

}

tail = tail->next; *// Di chuyển tail*

}

*// Nối phần còn lại*

tail->next = (a != NULL) ? a : b;

return dummy.next; *// Bỏ qua node giả*

}

**Test Case 1:**

TRON HAI DANH SACH DA SAP XEP

Nhap so luong phan tu cho danh sach A: 3

Nhap 3 phan tu cho danh sach A: 1 3 5

Nhap so luong phan tu cho danh sach B: 3

Nhap 3 phan tu cho danh sach B: 2 4 6

TRUOC KHI TRON

Danh sach A (da sap xep): 1 3 5

Danh sach B (da sap xep): 2 4 6

SAU KHI TRON

Danh sach sau khi tron: 1 2 3 4 5 6

Danh sach ket qua da duoc sap xep tang dan.

KIEM TRA NODE

Danh sach A sau tron:

Danh sach B sau tron:

Cac node da duoc tai su dung, khong con node roi.

**Test Case 2:**

Nhap so luong phan tu cho danh sach A: 4

Nhap 4 phan tu cho danh sach A: 5 2 8 1

Nhap so luong phan tu cho danh sach B: 3

Nhap 3 phan tu cho danh sach B: 4 7 3

Danh sach A chua sap xep. Dang sap xep...

Danh sach B chua sap xep. Dang sap xep...

TRUOC KHI TRON

Danh sach A (da sap xep): 1 2 5 8

Danh sach B (da sap xep): 3 4 7

SAU KHI TRON

Danh sach sau khi tron: 1 2 3 4 5 7 8